THE UNIVERSITY OF MELBOURNE

COMP 90048
Declarative Programming
Workshop 9 (week10)

2019 semester 1

by Wendy Zeng

Tutorial : Tue 18:15 - 19:15 221 Bouverie St, room B11

Wed 17:15 - 18:15 201 Bouverie St, room B132

Outline

1. Higher order predicates
2. All solutions

   2a. Setof and Bagof

   2b. Making Use of the Backtracking Features

3. Constraint Programming
4. More on Arithmetic and prolog problem solving (not in slides)

1. Higher Order Predicate

- Predicates that take other predicates as arguments

- call/1: call(:Goal)
  - to invoke prolog predicate dynamically
  - Example: call(write(“hello”)).
  - Example: X = write(“hello”), call(X).
- call/N: call(:Goal, Arg1, Arg2 …)
  - Partial application for :Goal, with Arg1, Arg2 … being the rest of the arguments
  - Example: call(plus(1), 2, X).
- apply/2: apply(:Goal, [Arg1, Arg2…])
  - Example: apply(plus(1), [2, X]).

1. Higher Order Predicate

- maplist/3: maplist(:Goal, List1, List2)
  - For each pair of element A and B with A from List1, B from List2, do call(:Goal, A, B)
  - Example: maplist(plus(1), [1,2,3], Result).
  - Example: head([E|_], E).
    maplist(head, [[1,2,3],[4,5,6],[7,8,9]], Result).
- maplist/2: maplist(:Goal, List)
  - For each element from List1 apply goal successively until the end of list of goal fails
  - Example: even_list(List) :- length(List, Len), Len mod 2 =:= 0.
    maplist(even_list,[[1,2],[3,4,5]]).

1. Higher Order Predicate

The SWI Prolog builtin predicate maplist/2 is analogous to the map/3 predicate defined in lectures, but with one less argument.

The query

sort([write('pear+'),write('apple')],A),maplist(call,A).

- Fails with no output.
- Produces output

  pear+apple

  then fails.
- Produces output

  pear+apple

  then succeeds with

  A = [write(apple), write('pear+')].
- Produces output

  applepear+

  then succeeds with

  A = [write(apple), write('pear+')].

- Single quote sign denotes atom in Prolog
- 'apple' is equivalent with apple in atomic form
- 'pear+' is not the same as pear+ so quotes remain to indicate the form as atom

- A: sort compound terms write('pear+') and write('apple') by rule of functor > arity > alphabetical order of corresponding arguments

- Maplist: perform call to each element in list equivalent to call(write('apple')), call(write('pear+')).

is true if Bag is the bag of all the instances of Template for which Goal is true. If no solutions are found it fails

2. All Solutions

2a. setof and bagof

- Group all solutions generated by Prolog backtracking mechanism into list of solutions
- bagof/3: bagof(+Template, :Goal, -Bag)
  - Backtracking on all free variables
  - succeeds if Bag is the list of all instances of Template for which the Goal holds true, if there is no solutions found then it fails
  - Example: bagof((Front, Back), append(Front, Back, [1,2,3]), Bag).
  - Example: bagof(Front, Back^append(Front, Back, [1,2,3]),Bag).
  - Example: bagof(X, (member(X, [1,2,2]), X > 1), Bag).

- setof/3: setof(+Template, :Goal, -Bag)
  - Bag list is sorted with no duplicates
  - Example: setof(X, (member(X, [1,2,2]), X > 1), Set).

2. All Solutions

With the facts

```
win(rock,scissors).
win(scissors,paper).
win(paper,rock).
```

The query

```
setof(W-L,win(W,L),S).
```

- Fails.
- Succeeds with

  S = [rock-scissors, scissors-paper, paper-rock].

- Succeeds with

  S = [paper-rock, rock-scissors, scissors-paper].

- Succeeds with

  S = rock-scissors ;
  S = scissors-paper ;
  S = paper-rock.

setof instantiate a list to contain all backtracked solutions and sort them by term comparison rule and remove duplicates

2. All Solutions

2b. Making Use of the Backing Features

- Example1: Path finding problem

  - A map is described as in the edge predicate
  - It describes all the direction that relates two points together
  - this predicate describes possible moves from one state to another

  - Each atom a, b, c … represents a ***state*** (the point of location)

```
a - b - c
    |   |
    d - e - f
    |   |   |
    g   h - i
```

```
edge(a,  east, b).
edge(b,  west, a).
edge(b, south, d).
edge(b,  east, c).
edge(c,  west, b).
edge(c, south, e).
edge(d, north, b).
edge(d,  east, e).
edge(d, south, g).
edge(e,  west, d).
edge(e, north, c).
edge(e,  east, f).
edge(e, south, h).
edge(f,  west, e).
edge(f, south, i).
edge(g, north, d).
edge(h, north, e).
edge(h,  east, i).
edge(i,  west, h).
edge(i, north, f).
```

2. All Solutions

2b. Making Use of the Backing Features

- Example1: to find all the possible paths that connect from a starting point to an ending point

```
a - b - c
    |   |
    d - e - f
    |   |   |
    g   h - i
```

```
path(Start, Start, [], _).
path(Start, End, [Move|Moves], Histories) :-
    edge(Start, Move, Next),
    \+member(Next, Histories),
    path(Next, End, Moves, [Next|Histories]).
```

Explore one move at a time, see what is the next possible point

Make sure that this next possible point has not been visited yet

Recursively generate the rest of the paths from this next possible point to the target point

2. All Solutions

2b. Making Use of the Backing Features

- Example2: Two containers problem

move(+Starting_state, -Next_state, +One_move)

```
move(([Big, _], [Small, S]), ([Big, 0], [Small, S]), empty(Big)).
move(([Big, B], [Small, _]), ([Big, B], [Small, 0]), empty(Small)).
move(([Big, _], [Small, S]), ([Big, Big], [Small, S]), fill(Big)).
move(([Big, B], [Small, _]), ([Big, B], [Small, Small]), fill(Small)).
move(([Big, B], [Small, S]), ([Big, B_new], [Small, S_new]), pour(Big,
Small)) :-
        pour(Small, B, S, B_new, S_new).
move(([Big, B], [Small, S]), ([Big, B_new], [Small, S_new]), pour(Small,
Big)) :-
        pour(Big, S, B, S_new, B_new).
```

2. All Solutions

2b. Making Use of the Backing Features

- Example2: Two containers problem
- State is represented as

([Big_capacity, Big_volume), ([Small_capacity, Small_volume)]

- empty, fill, pour are possible movements that can transit from one state to another
- The move predicates describes the starting and ending state that each of these movement act on

2. All Solutions

2b. Making Use of the Backing Features

- Example2: Get 4 L of water in the big container and both start with empty

```
explore_moves(State0, State0, [], _).
explore_moves(State_current, State_final, [Current_move|Other_moves], State_logs)
:-
    move(State_current, State_next, Current_move),
    \+member(State_next, State_logs),
    explore_moves(State_next, State_final, Other_moves, [State_next|State_logs]).

containers(Moves) :- explore_moves(([5, 0], [3, 0]), ([5, 4], [3, _]), Moves, [([5, 0],
[3, 0])]).
```

3. Constraint Programming

Finite Domain Constraints:

```
sudoku(Rows) :-
length(Rows, 9), maplist(same_length(Rows), Rows),
append(Rows, Vs), Vs ins 1..9,
maplist(all_distinct, Rows),
transpose(Rows, Columns),
maplist(all_distinct, Columns),
Rows = [A,B,C,D,E,F,G,H,I],
blocks(A, B, C), blocks(D, E, F), blocks(G, H, I).
blocks([], [], []).
blocks([A,B,C|Bs1], [D,E,F|Bs2], [G,H,I|Bs3]) :-
all_distinct([A,B,C,D,E,F,G,H,I]),
blocks(Bs1, Bs2, Bs3).
```

- A finite set of variables over a domain
  - X ins 1..9
- A finite set of constraints:
  - consistency constraint:
    - All row/col/block same length
  - Global constraint:
    - No duplicates in row/col/ block
- Search techniques:
  - Backtracking

THE UNIVERSITY OF MELBOURNE

Thank you

wendy.zeng@unimelb.edu.au

By Wendy Zeng